Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа №2 по курсу**

**«Операционные системы»**

**Процессы операционных систем**

Студент: [ Чубуков Андрей Владимирович]

Группа: М8О–208Б–21

Вариант: 14

Преподаватель: Соколов Андрей Алексеевич

Оценка: \_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_

Москва, 2022.

**Постановка задачи**

## Цель работы

Целью является приобретение практических навыков в:

* Управление процессами в ОС
* Обеспечение обмена данных между процессами посредством каналов

## Задание

Составить и отладить программу на языке Си, осуществляющую работу с процессами и взаимодействие между ними в одной из двух операционных систем. В результате работы программа (основной процесс) должен создать для решение задачи один или несколько дочерних процессов. Взаимодействие между процессами осуществляется через системные сигналы/события и/или каналы (pipe). Необходимо обрабатывать системные ошибки, которые могут возникнуть в результате работы

**Общие сведения о программе**

Реализация программы была бы невозможна без специальной библиотеки “unistd.h” для операционной системы Linux, которая позволяет работать с процессами и системными вызовами.  
По мере реализации задания используются такие строки(команды), как:  
**int fd[2]** - создание массива из 2 дескрипторов, 0 - чтение (read), 1 - передача (write)

**pipe(fd)** - конвейер, с помощью которого выход одной команды подается на вход другой (оно же “труба”)

**int id = fork ()** - создание дочернего процесса, в переменной id будет лежать “специальный код” процесса (-1 - ошибка fork, 0 - дочерний процесс, >0 - родительский)  
  
**read(...)** - команда, предназначенная для чтения данных, посланных из другого процесса, принимающая на вход три параметра: элемент массива дескрипторов с индексом 0, значение **получаемого** объекта (переменной, массива и т.д.), размер **получаемого** объекта (например, в случае переменной int - sizeof(int), в случае массива из 10 переменных типа int - sizeof(int) \* 10)

**write(...) -** команда, принимающая на вход три параметра: элемент массива дескрипторов с индексом 1, значение **посылаемого** объекта (переменной, массива и т.д.), размер **посылаемого** объекта (например, в случае переменной int - sizeof(int), в случае массива из 10 переменных типа int - sizeof(int) \* 10)  
  
**close(...)** - команда, использующаяся, когда нам больше не нужно передавать, либо считывать что-либо из другого процесса.

**Общий метод и алгоритм решения**.

Для реализации поставленной задачи необходимо:

1. Изучить принципы работы unistd.h.
2. В файле Lab1-2.c содержится программа, получающая на вход строку произвольной длинны.
3. Строка подается в первый процесс Child1, который переводит строки в нижный регистр, и через pipe() передает строку в Child2.
4. Child2 получает строку и убирает все двойные пробелы, передает строку в материнский процесс.
5. Материнский процесс выводит строку в стандартный поток вывода.

**Основные файлы программы**

**Lab1-2.c:**

**#include "unistd.h"**

#include "stdio.h"

#include "string.h"

#include "stdlib.h"

int main()

{

int fd1[2];

int fd2[2];

int fd3[2];

pipe(fd1);

int tr = 0;

pipe(fd2);

pipe(fd3);

char s[100];

int n =0;

printf("Enter text (} - sign of the end):\n");

for (int i =0; i<=100;i++)

{

scanf("%c", &s[i]);

if (s[i]!='}')

{

n++;

}else

{

break;

}

}

write(fd1[1],&n,sizeof(int));

write(fd1[1],s,sizeof(char)\*n);

int child1=fork();

if (child1 == -1)

{

printf("fork 1 error\n");

return -1;

} else if (child1 == 0 )

{

printf("child 1 is working\n");

int a;

read(fd1[0], &a, sizeof(int));

char stringAr[a];

read(fd1[0], stringAr, sizeof(char)\*a);

for (int i =0;i<a;i++)

{

if ((stringAr[i] >64) &&(stringAr[i] < 91))

{

stringAr[i] = stringAr[i]+32;

}

}

write(fd2[1],&a,sizeof(int));

write(fd2[1],stringAr,sizeof(char)\*a);

/

}else{

int child2=fork();

if (child2 == -1)

{

printf("fork 2 error\n");

return -1;

} else if (child2 == 0)

{

printf("child 2 is working\n");

int a1;

int n1 =0;

read(fd2[0],&a1,sizeof(int));

char stringAr1[a1];

read(fd2[0], stringAr1, sizeof(char)\*a1);

for (int i =0;i<a1;i++)

{

if ((stringAr1[i] == 32) &&(stringAr1[i+1] == 32))

{

for (int j = i;j<a1-1;j++)

{

stringAr1[j]=stringAr1[j+1];

}

n1++;

}

}

a1=a1-n1;

close(fd2[0]);

close(fd2[1]);

write(fd3[1],&a1,sizeof(int));

write(fd3[1],stringAr1,sizeof(char)\*a1);

}

if (child2 >0)

{

int final;

read(fd3[0],&final,sizeof(int));

char finalString[final];

read(fd3[0],finalString,sizeof(char)\*final);

for(int i =0;i<final;i++)

{

printf("%c", finalString[i]);

}

printf("\n");

close(fd3[0]);

close(fd3[1]);

}

}

}

**Пример работы**

**№1**

andrey@andrey-HP:~/OS$ gcc Lab1-2.c -o l1

andrey@andrey-HP:~/OS$ ./l1

Enter text (} - sign of the end):

QWEwe we qwe} ← Ввел данные с клавиатуры, верхний регистр.

child 1 is working

child 2 is working

qwewe we qwe

**№2**

andrey@andrey-HP:~/OS$ gcc Lab1-2.c -o l1

andrey@andrey-HP:~/OS$ ./l1

Enter text (} - sign of the end):

234гнуцкншг32 уцшк} ←-Ввел с клавиатуры, два пробела

child 1 is working

child 2 is working

234гнуцкншг32 уцшк

**№3**

andrey@andrey-HP:~/OS$ gcc Lab1-2.c -o l1

andrey@andrey-HP:~/OS$ ./l1

Enter text (} - sign of the end):

QWEqwre 1234 2 we qwrQ 2}

child 1 is working

child 2 is working

qweqwre 1234 2 we qwrq 2

**Вывод**

Данная лабораторная работа помогла мне ознакомиться с тем, как устроены процессы в Linux. Я осознал принцип работы вышеперечисленных команд и системных вызовов, а также узнал некоторые тонкости работы процессоров.